



Pict correctness revisited^{☆,☆☆}

Philippe Bidinger^{a,*}, Adriana Compagnoni^b

^a VERIMAG, Grenoble, France

^b Stevens Institute of Technology, Hoboken, NJ, USA

ARTICLE INFO

Keywords:

Pi-calculus
Abstract machine
Fairness
Pict

ABSTRACT

The Pict programming language is an implementation of the π -calculus in which executions of π -calculus terms are specified via an abstract machine. An important property of any concurrent programming language implementation is the fair execution of threads. After defining fairness for the π -calculus, we show that Pict abstract machine executions implement fair π -calculus executions. We also give new proofs of soundness and liveness for the Pict abstract machine.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

The π -calculus [14,17] is a minimal language designed to capture and model key concepts of communicating concurrent systems in a formal setting. It emphasizes channel-based communication, dynamic channel creation and the ability to communicate channels as data. Pict [19,16] is a high-level programming language purely based on π -calculus primitives. Pict's runtime environment is based on a formal abstract machine specification, but little emphasis has been placed on its correctness.

The correctness of a programming language runtime is critical since, in order to be able to reason about programs, we need the guarantee that programs are executed according to their semantics. Correctness results of implementations usually relate executions of terms in a high-level language to its implementation in a low-level language. The low-level language can be an existing process calculus or, like in Pict, an abstract machine specification.

In recent years, many process calculi based on the π -calculus have been introduced to study the dynamics of existing or new paradigms of computation, such as distributed computing, global computing, or component-based programming. Much work has been done on the distributed implementation of these calculi [5,7,22,20,10,8,11,12,1]. On the other hand, since the definition of Pict, there has been no new insight for the local implementation of these calculi. Therefore, Pict is still a reference implementation of the π -calculus, and we think that proving its correctness is a first step toward more general proofs of correctness of implementations of these calculi.

The π -calculus is a concurrent language where concurrency is modeled using a non-deterministic reduction relation. The Pict Abstract Machine (PAM) implements a particular scheduling strategy that corresponds to a subset of the possible executions in the π -calculus. It is therefore impossible to state an exact correspondence between π -calculus executions and PAM executions. Instead, we will prove the correctness of the abstract machine with three properties:

- A soundness property that states that PAM executions correspond to valid π -calculus executions.
- A liveness property that ensures that the abstract machine is not stuck when its state corresponds to a π -calculus term that can reduce.
- A fairness property that characterizes PAM executions among possible π -calculus executions.

[☆] This work was funded in part by the US Army under contract W15QKN-05-D-0011.

^{☆☆} A short version of this paper appeared in Proceedings of FMOODS 2007.

* Corresponding author. Tel.: +33 4 56 52 04 15.

E-mail address: philippe.bidinger@imag.fr (P. Bidinger).

These properties are fairly standard but have not been proven for Pict yet (see Section 5 for details). The main contribution of this paper lies in the statement and proof of a fairness property for Pict. To our knowledge, no implementation of a process calculus has been proven fair so far, although fairness is conjectured in [21,19]. Moreover, the technique we propose is general enough to be adapted to similar settings.

Informally, we say that an execution is *weakly fair* if a prefixed process able to communicate *continuously* will eventually do so. An execution is *strongly fair* if a prefixed process able to communicate *infinitely often* will eventually do so. Consider for instance the π -calculus term

$$x!(a) \mid *x?(z).x!(z) \mid y!(0) \mid y?(z).0$$

where $*x?(z).P$ represents replicated input. There are valid infinite executions in which the communication on y never takes place even though at any time this communication is possible. Similarly, in the term

$$x!(a) \mid *x?(z).y!(z) \mid *y?(z).x!(z) \mid *x?(z).x!(z)$$

there are infinite executions in which the communication on the last process never takes place, even though such a communication might happen infinitely often (but not continuously). The intuitive expectation of a programmer is that all processes running in parallel will be interleaved *fairly* and so such executions are considered unsatisfactory.

Stating a fairness property for the π -calculus is not immediate. The definition of the π -calculus makes it difficult to identify subprocesses within a process and, in particular, it is difficult to state properties about fair executions of these processes. When considering π -calculus processes, mainly two kinds of confusion can arise.

- Processes are identified up to renaming of bound names and lead to possible confusion of channels. For instance, we have

$$\nu x.vy.x!() \mid y!() \mid R \rightarrow \nu x.vy.x!() \mid R'.$$

Because of possible renamings of x and y , we do not know which channel reacted.

- Confusion of structurally equivalent processes.

$$P = x!() \mid x?().x!() \mid *x?().x!()$$

$$P' = x!() \mid x?().x!() \mid x?().x!() \mid *x?().x!().$$

We have $P \equiv P'$ and $P' \rightarrow P$, and we do not know which receivers react with $x!()$.

A possible solution is to define an auxiliary calculus in which prefixes are annotated with *labels* in such a way that labels uniquely denote prefixes and that this property is invariant throughout reduction [4,3]. A *live action* of a term is then defined as a couple of labels corresponding to prefixed processes that can react. An infinite labeled execution is strongly fair if there are no labels appearing in an infinity of live actions.

This paper is organized as follows. In Section 2, we define the π -calculus and its labeled variant that allows us to define strong fairness. In Section 3, we define the Pict abstract machine, and a labeled version. We discuss the correctness of the abstract machine in Section 4. In Section 5, we discuss related work. We conclude with future research directions in Section 6.

2. Fairness in the π -calculus

2.1. The π -calculus

We suppose given a set of names N ranged over by x, y, \dots . We define the set of π -calculus processes P as follows:

$$P, Q, \dots ::= 0 \mid \pi.P \mid \nu x.P \mid P \mid P$$

$$\pi ::= x!(y) \mid x?(y) \mid *x?(y).$$

The π -calculus evaluation contexts are given by:

$$\mathbf{E} ::= \cdot \mid \nu x.\mathbf{E} \mid P \mid \mathbf{E} \mid \mathbf{E} \mid P.$$

The operational semantics is defined as the smallest relation such that rules in Fig. 2 hold. It makes use of a structural equivalence relation defined as the smallest equivalence relation satisfying the rules in Fig. 1. As usual, $\text{fn}(P)$ denotes the set of free names of process P , and $=_\alpha$ equates two processes that differ only by their bound names. We write $\mathbf{E}[P]$ for the context \mathbf{E} in which the hole \cdot has been substituted with P .

Without loss of generality, we restrict the usual replication operator to input processes. Rule R.REP models communication with a replicated input process.

2.2. A labeled π -calculus

Informally, a fair execution of a process is an execution in which no subprocess is ready to participate in a communication infinitely often. To formalize this statement, we need to identify in a process the subprocesses that can participate in a communication, and keep track of their identities throughout reductions.

$$\begin{array}{l}
(P \mid Q) \mid R \equiv P \mid (Q \mid R) \text{ S.PAR.ASSOC} \qquad P \mid Q \equiv Q \mid P \text{ S.PAR.COM} \\
P \mid \mathbf{0} \equiv P \text{ S.PAR.NIL} \qquad \nu x.\mathbf{0} \equiv \mathbf{0} \text{ S.NU.NIL} \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P \text{ S.NU.COM} \\
\frac{x \notin \text{fn}(Q)}{(\nu x.P) \mid Q \equiv \nu x.P \mid Q} \text{ S.NU.PAR} \qquad \frac{P =_{\alpha} Q}{P \equiv Q} \text{ S.}\alpha \qquad \frac{P \equiv Q}{\mathbf{E}[P] \equiv \mathbf{E}[Q]} \text{ S.CTX}
\end{array}$$

Fig. 1. Structural equivalence.

$$\begin{array}{l}
\frac{}{x!(y).P \mid x?(z).Q \rightarrow P \mid Q\{y/z\}} \text{ R.RED} \\
\frac{}{x!(y).P \mid *x?(z).Q \rightarrow P \mid Q\{y/z\} \mid *x?(z).Q} \text{ R.REP} \qquad \frac{P \rightarrow Q}{\mathbf{E}[P] \rightarrow \mathbf{E}[Q]} \text{ R.CTX} \\
\frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q} \text{ R.STR}
\end{array}$$

Fig. 2. Reduction relation.

$$\begin{array}{l}
\frac{}{\text{wf}(\mathbf{0}, \mathcal{L})} \text{ WF.NIL} \qquad \frac{\text{lab}(\overline{P}) \uplus \text{lab}(\overline{P}') \subseteq \mathcal{L}}{\text{wf}(\overline{P} \mid \overline{P}', \mathcal{L})} \text{ WF.PAR} \qquad \frac{\text{wf}(\overline{P}, \mathcal{L})}{\text{wf}(\nu x.\overline{P}, \mathcal{L})} \text{ WF.NEW} \\
\frac{\text{wf}(\overline{P}, \mathcal{L})}{\text{wf}(\pi_l.\overline{P}, \mathcal{L} \uplus \{l\})} \text{ WF.PREFIX}
\end{array}$$

Fig. 3. Well-formed labeled process.

To do so, we follow [4,3] and define a labeled version of the π -calculus in which prefixes are annotated with labels. A label has to identify a prefix uniquely in an entire execution of a process. In other words, not only do prefixes have distinct labels in a process, but when new prefixes are created, their labels are new with respect to all the labels in the past execution of the term. We then characterize the labels belonging to prefixes that can participate in a communication. Finally, we give the definition of fairness.

We denote by L a set of labels such that $L \cap N = \emptyset$. We use $\mathcal{P}^f(L)$ to denote the finite subsets of L . A labeled process is a pair made of a π -calculus process in which prefixes are labeled, and a finite set of labels. The set of labeled processes LP is generated by the grammar given below.

$$\begin{array}{l}
C, D, \dots ::= \overline{P}, \mathcal{L} \\
\overline{P}, \overline{Q}, \dots ::= \mathbf{0} \mid \pi_l.\overline{P} \mid \nu x.\overline{P} \mid \overline{P} \mid \overline{P} \\
l \in L \\
\mathcal{L} \in \mathcal{P}^f(L).
\end{array}$$

We also extend contexts with labels and we denote labeled contexts $\overline{\mathbf{E}}$.

We need several auxiliary functions. The function lab returns the set of all labels of a process or a context. The function unl erases all labeling information from a labeled process.

In order to ensure that labels occur uniquely in a process, we define a well-formation predicate wf as the smallest relation on LP such that rules in Fig. 3 hold. We write $A \uplus B$ for $A \cup B$ when $A \cap B = \emptyset$. A labeled process C is said to be well-formed if we have $\text{wf}(C)$. We denote by WFP the set of well-formed labeled processes.

The operational semantics is defined in the same way as for the π -calculus via a structural equivalence relation \equiv and a reduction relation \rightarrow , both binary relations over LP . The structural equivalence is defined, as before, as the smallest equivalence relation that verifies rules in Fig. 1 (where prefixes are labeled and equivalent processes have the same set of labels). The reduction relation is the smallest relation that verifies the rules in Fig. 4. The main difference with the unlabeled reduction relation appears in the rule LR.REP for replicated input in which fresh labels are generated.

Labeling is stable under reduction and structural equivalence. Hence, in the following, we consider only well-formed processes.

Lemma 1 (Stability of Labeling). (i) If $C \equiv D$ and $C \in WFP$, then $D \in WFP$.
(ii) If $C \rightarrow D$ and $C \in WFP$, then $D \in WFP$.

The following lemma shows that the labeling system has been designed so that no label can occur more than once in a labeled term, and once a label disappears, it does not reappear in the system.

Lemma 2 (Uniqueness of Labeling). (i) If $C \in WFP$ then no label l occurs more than once in C .
(ii) If $C \in WFP$, $C \rightarrow^* C' \rightarrow^* C''$ and $l \in \text{lab}(C) \cap \text{lab}(C'')$, then $l \in \text{lab}(C')$.

$$\begin{array}{c}
\frac{}{x!(y)_l.\bar{P} \mid x?(z)_{l'}.\bar{Q}, \mathcal{L} \rightarrow \bar{P} \mid \bar{Q}\{y/z\}, \mathcal{L}} \text{LR.RED} \\
\\
\frac{\alpha \text{ injective and } \mathcal{L}' = \mathcal{L} \uplus \alpha(\mathcal{L})}{x!(y)_l.\bar{P} \mid *x?(z)_{l'}.\bar{Q}, \mathcal{L} \rightarrow \bar{P} \mid \bar{Q}\{y/z\} \mid \alpha(*x?(z)_{l'}.\bar{Q}), \mathcal{L}'} \text{LR.REP} \\
\\
\frac{\bar{P}, \mathcal{L} \rightarrow \bar{P}', \mathcal{L}' \quad \text{lab}(\bar{E}) \subseteq \mathcal{L}}{\bar{E}[\bar{P}], \mathcal{L} \rightarrow \bar{E}[\bar{P}'], \mathcal{L}'} \text{LR.CTX} \quad \frac{D \equiv D' \quad D' \rightarrow C' \quad C' \equiv C}{D \rightarrow C} \text{LR.STR}
\end{array}$$

Fig. 4. Labeled reduction relation.

The labeled π -calculus is a conservative extension of the π -calculus. A labeled process has exactly the same reductions as the corresponding unlabeled process. Moreover, we can label any process into a well-formed labeled process

Proposition 1 (Operational Correspondence). *Let $P \in \mathbf{P}$ and $C \in \mathbf{WFP}$ such that $P = \text{unl}(C)$. We have*

- (i) $P \rightarrow P'$ implies $\exists C' \in \mathbf{LP}$ such that $C \rightarrow C'$ and $\text{unl}(C') = P'$.
- (ii) $C \rightarrow C'$ implies $P \rightarrow \text{unl}(C')$.

Proposition 2 (Existence of a Labeling). *For all $P \in \mathbf{P}$ there exists $C \in \mathbf{WFP}$ such that $\text{unl}(C) = P$.*

We now define the live actions of a labeled process. A live action is a pair of labels corresponding to prefixed processes that can immediately react.

Definition 1 (Live Actions). The set of live actions of a labeled process $C = \bar{P}, \mathcal{L}$ is defined as

$$LA(C) = \{[l, l'] / C \equiv v\tilde{x}.y!(v)_l.\bar{P}_0 \mid y?(z)_{l'}.\bar{P}_1 \mid \bar{P}_2, \mathcal{L} \text{ or } C \equiv v\tilde{x}.y!(v)_l.\bar{P}_0 \mid *y?(z)_{l'}.\bar{P}_1 \mid \bar{P}_2, \mathcal{L}\}.$$

We also define the set of labels belonging to a live action as

$$L(C) = \{l \in x/x \in LA(C)\}.$$

The following lemma states a correspondence between live actions and reductions.

Lemma 3. $C \rightarrow C'$ for some C' if and only if $LA(C) \neq \emptyset$.

Definition 2 (Execution). For an arbitrary relation \rightarrow , an execution is a sequence of terms T_0, T_1, \dots , possibly infinite, such that $T_0 \rightarrow \dots \rightarrow T_n \rightarrow \dots$.

We can now define a strong fairness property for the labeled calculus. An execution is fair if a prefix cannot potentially participate in a reduction infinitely often. According to this definition, we only need to consider infinite executions.

Definition 3 (Strong Fairness in the Labeled π -calculus). An infinite execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ is fair if for any strictly increasing sequence of integers $(u_n)_{n \in \mathbb{N}}$, we have $\bigcap_{n \in \mathbb{N}} L(C_{u_n}) = \emptyset$.

An execution in the π -calculus is fair if any corresponding labeled execution is fair.

Definition 4 (Strong Fairness in the π -calculus). An infinite execution $P_0 \rightarrow \dots \rightarrow P_n \rightarrow \dots$ is fair if any execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ such that $\forall i \in \mathbb{N}. \text{unl}(C_i) = P_i$ is fair.

In fact, as a consequence of the next lemma, we can use a weaker definition. An execution is fair if there exists a corresponding fair labeled execution.

Lemma 4. *Let $P_0 \rightarrow \dots \rightarrow P_n \rightarrow \dots$ be an infinite execution. The two following propositions are equivalent:*

- (i) *There exists a fair labeled execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ such that $\text{unl}(C_i) = P_i$.*
- (ii) *Any labeled execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ such that $\text{unl}(C_i) = P_i$ is fair.*

3. Abstract machine

3.1. Syntax and operational semantics

The syntax of the Pict abstract machine is given in Fig. 5 and follows closely [19].¹ A machine state, or PAM term, consists of a queue of π -calculus processes \mathcal{P} (the runqueue), a heap \mathcal{H} and a set of names \mathcal{N} . A heap is a function that maps channel names to process queues. We denote by \mathbf{M} the set of machine states. We often omit the set of names \mathcal{N} in PAM terms when it is not important, in particular in reduction rules where it remains unchanged. We also write $\mathcal{P} :: \mathcal{Q}$ for the appending of \mathcal{P} and \mathcal{Q} . The operational semantics is defined via two reduction relations, defined as the smallest binary relations over machine states that satisfy the inference rules given in Fig. 6. Intuitively, the relation \rightsquigarrow corresponds to the implementation of \equiv , whereas \mapsto implements the actual communication. An actual implementation of this abstract machine does not need to distinguish these relations and would implement $\rightarrow = \rightsquigarrow \uplus \mapsto$, but this distinction will help us to prove correctness

¹ In particular, this presentation makes use of synchronous communications.

$\mathcal{M} ::= \langle \mathcal{P}, \mathcal{H}, \mathcal{N} \rangle$	State
$\mathcal{P}, \mathcal{Q}, \dots ::= [] \mid P :: \mathcal{P}$	Processes Queue
$\mathcal{H} ::= \{x \rightarrow \mathcal{P}_x\}_{x \in \mathbb{N}}$	Heap

Fig. 5. Syntax of PAM terms.

$$\begin{array}{c}
\frac{}{\langle \mathbf{0} :: \mathcal{Q}, \mathcal{H} \rangle \rightsquigarrow \langle \mathcal{Q}, \mathcal{H} \rangle} \text{AM.NIL} \qquad \frac{}{\langle (P \mid Q) :: \mathcal{Q}, \mathcal{H} \rangle \rightsquigarrow \langle P :: \mathcal{Q} :: Q, \mathcal{H} \rangle} \text{AM.PAR} \\
\\
\frac{z = \text{freshn}(\mathcal{N})}{\langle \nu x.P :: \mathcal{Q}, \mathcal{H}, \mathcal{N} \rangle \rightsquigarrow \langle P\{z/x\} :: \mathcal{Q}, \mathcal{H}, \mathcal{N} \uplus \{z\} \rangle} \text{AM.NEW} \\
\\
\frac{\mathcal{P} = [] \vee \mathcal{P} = x?(z).Q :: \mathcal{P}' \vee \mathcal{P} = *x?(z).Q :: \mathcal{Q}}{\langle x?(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \rightsquigarrow \langle \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P} :: x?(y).P \} \rangle} \text{AM.PUSHRECEIVER} \\
\\
\frac{\mathcal{P} = [] \vee \mathcal{P} = x!(z).Q :: \mathcal{P}'}{\langle x!(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \rightsquigarrow \langle \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P} :: x!(y).P \} \rangle} \text{AM.PUSHMESSAGE} \\
\\
\frac{\mathcal{P} = [] \vee \mathcal{P} = x?(z).Q :: \mathcal{P}' \vee \mathcal{P} = *x?(z).Q :: \mathcal{P}'}{\langle *x?(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \rightsquigarrow \langle \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P} :: *x?(y).P \} \rangle} \text{AM.PUSHREPRECEIVER} \\
\\
\frac{\mathcal{P} = x!(z).Q :: \mathcal{P}'}{\langle x?(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \mapsto \langle P\{z/y\} :: \mathcal{Q} :: Q, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}' \} \rangle} \text{AM.COM1} \\
\\
\frac{\mathcal{P} = x!(z).Q :: \mathcal{P}'}{\langle *x?(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \mapsto \langle *x?(y).P :: \mathcal{Q} :: P\{z/y\} :: Q, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}' \} \rangle} \text{AM.RCOM1} \\
\\
\frac{\mathcal{P} = x?(z).Q :: \mathcal{P}'}{\langle x!(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \mapsto \langle P :: \mathcal{Q} :: Q\{y/z\}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}' \} \rangle} \text{AM.COM2} \\
\\
\frac{\mathcal{P} = *x?(z).Q :: \mathcal{P}'}{\langle x!(y).P :: \mathcal{Q}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}\} \rangle \mapsto \langle P :: \mathcal{Q} :: Q\{y/z\}, \mathcal{H} \oplus \{x \rightarrow \mathcal{P}' :: *x?(z).Q \} \rangle} \text{AM.RCOM2}
\end{array}$$

Fig. 6. PAM reduction rules.

properties. In rule AM.NEW, we suppose there is a function $\text{freshn} : \mathcal{P}^f(\mathbb{N}) \rightarrow \mathbb{N}$ such that $\text{freshn}(\mathcal{N}) \notin \mathcal{N}$. We also suppose that names generated by the freshn function never appear in the π -calculus processes in the PAM term (this could be enforced by defining a new syntactic category of names).

We refer the reader to [19,16] for detailed explanations of these rules. We briefly summarize the main ideas here. An execution of the abstract machine starts with an empty heap (we denote it with $\mathcal{H}_{[]}$) that maps all names to empty queues of processes, and a runqueue containing the π -calculus process to be executed. Depending on the form of the process at the top of the runqueue, and the state of the heap, exactly one rule can apply. The execution stops when the runqueue is empty.

A nil process is discarded from the runqueue (rule AM.NIL). Parallel composition of processes is split into two processes. The left hand process is kept at the head of the runqueue, while the right hand process is pushed to the end (rule AM.PAR). Rule AM.NEW implements name restriction by generating new fresh names. When the first term of the runqueue is a prefixed process willing to communicate on a name x , there are two possible cases. If there is no corresponding process in the heap, the process is pushed on the heap queue for x (rules AM.PUSHMESSAGE, AM.PUSHRECEIVER, AM.PUSHREPRECEIVER). If there is a corresponding process in the heap queue (the first element), the communication is performed and the continuation of the receiver and sender are placed in the runqueue (rules AM.COM1, AM.RCOM1, AM.COM2, AM.RCOM2).

Remark 1. In the following, we will consider only machine terms \mathcal{M} for which there exists a process P such that $\langle P :: [], \mathcal{H}_{[]}, \emptyset \rangle \rightarrow^* \mathcal{M}$. In particular, we can show that processes appearing in an association $x \rightarrow \mathcal{P}$ are of the form $\pi.P$, where all prefixes are either output on x , or input (replicated or not) on x . We can also show that \mathcal{H} is finite. Moreover, we can notice that the relation \rightarrow is deterministic. In particular, generated fresh names are fully determined by the function freshn in rule AM.NEW.

3.2. Labeled abstract machine

We define a labeled version of the Pict abstract machine and essentially follow Section 2. This auxiliary calculus is a technical tool, and it is only used for proving the correctness of the abstract machine. Its syntax is defined by adding labels

$$\begin{array}{c}
\frac{\text{wf}(\overline{\mathcal{P}}) \quad \text{wf}(\overline{\mathcal{H}}) \quad \text{lab}(\overline{\mathcal{P}}) \uplus \text{lab}(\overline{\mathcal{H}}) \subseteq \mathcal{L}}{\text{wf}(\langle \overline{\mathcal{P}}, \overline{\mathcal{H}}, \mathcal{N}, \mathcal{L} \rangle)} \text{WF.STATE} \\
\\
\frac{\text{wf}(\overline{\mathcal{P}}) \quad \text{wf}(\overline{\mathcal{P}}) \quad \text{lab}(\overline{\mathcal{P}}) \cap \text{lab}(\overline{\mathcal{P}}) = \emptyset}{\text{wf}(\overline{\mathcal{P}} :: \overline{\mathcal{P}})} \text{WF.PROCQUEUE} \\
\\
\frac{\forall x, y \in \mathbb{N}. x \neq y \implies \text{lab}(\overline{\mathcal{H}}(x)) \cap \text{lab}(\overline{\mathcal{H}}(y)) = \emptyset \quad \forall x \in \mathbb{N}. \text{wf}(\overline{\mathcal{H}}(x))}{\text{wf}(\overline{\mathcal{H}})} \text{WF.HEAP}
\end{array}$$

Fig. 7. Well-formed PAM term.

$$\begin{array}{c}
\frac{\mathcal{P} = x!(z)_{l'}. \overline{Q} :: \overline{\mathcal{P}}' \quad \alpha \text{ injective and } \mathcal{L}' = \mathcal{L} \uplus \alpha(\mathcal{L})}{\langle *x?(y)_l. \overline{P} :: \overline{Q}, \overline{\mathcal{H}} \oplus \{x \rightarrow \overline{\mathcal{P}}\}, \mathcal{L} \rangle \mapsto \langle \alpha(*x?(y)_l. \overline{P}) :: \overline{Q} :: \overline{P}\{z/y\} :: \overline{Q}, \overline{\mathcal{H}} \oplus \{x \rightarrow \overline{\mathcal{P}}'\}, \mathcal{L}' \rangle} \text{AM.RCOM1} \\
\\
\frac{\mathcal{P} = *x?(z)_{l'}. \overline{Q} :: \overline{\mathcal{P}}' \quad \alpha \text{ injective and } \mathcal{L}' = \mathcal{L} \uplus \alpha(\mathcal{L})}{\langle x!(y)_l. \overline{P} :: \overline{Q}, \overline{\mathcal{H}} \oplus \{x \rightarrow \overline{\mathcal{P}}\}, \mathcal{L} \rangle \mapsto \langle \overline{P} :: \overline{Q} :: \overline{Q}\{y/z\}, \overline{\mathcal{H}} \oplus \{x \rightarrow \overline{\mathcal{P}}' :: \alpha(*x?(z)_{l'}. \overline{Q})\}, \mathcal{L}' \rangle} \text{AM.RCOM2}
\end{array}$$

Fig. 8. Labeled PAM reduction rules.

to π -calculus processes appearing in PAM terms. We also extend PAM terms with a finite set of labels. We write LM for the set of labeled PAM terms, and we use $\overline{\mathcal{M}}$ and its variants to range over them.

$$\begin{array}{ll}
\overline{\mathcal{M}} ::= \langle \overline{\mathcal{P}}, \overline{\mathcal{H}}, \mathcal{N}, \mathcal{L} \rangle & \text{State} \\
\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \dots ::= [] \mid \overline{P} :: \overline{\mathcal{P}} & \text{Processes Queue} \\
\overline{\mathcal{H}} ::= \{x \rightarrow \overline{\mathcal{P}}_x\}_{x \in \mathbb{N}} & \text{Heap} \\
\mathcal{L} \in \mathcal{P}^f(\mathbb{L}). &
\end{array}$$

We define the set of well-formed PAM terms in Fig. 7 and call it WFM. Reduction of labeled PAM terms is defined almost exactly as in the unlabeled calculus, apart from the rules AM.RCOM1 and AM.RCOM2 given in Fig. 8. The functions lab and unl extend as expected on process queues, heaps and machine states.

Lemma 5 (Stability of Labeling). (i) If $\overline{\mathcal{M}} \equiv \overline{\mathcal{M}}'$ and $\overline{\mathcal{M}} \in \text{WFM}$, then $\overline{\mathcal{M}}' \in \text{WFM}$.
(ii) If $\overline{\mathcal{M}} \rightarrow \overline{\mathcal{M}}'$ and $\overline{\mathcal{M}} \in \text{WFM}$, then $\overline{\mathcal{M}}' \in \text{WFM}$.

Proposition 3 (Operational Correspondence). Let $\mathcal{M} \in \mathbb{M}$ and $\overline{\mathcal{M}} \in \text{WFM}$ such that $\mathcal{M} = \text{unl}(\overline{\mathcal{M}})$. If \Rightarrow denotes either \mapsto or \rightsquigarrow , we have

- (i) $\mathcal{M} \Rightarrow \mathcal{M}'$ implies there is $\overline{\mathcal{M}}' \in \text{LM}$ such that $\overline{\mathcal{M}} \Rightarrow \overline{\mathcal{M}}'$ and $\text{unl}(\overline{\mathcal{M}}') = \mathcal{M}'$.
- (ii) $\mathcal{M} \Rightarrow \overline{\mathcal{M}}'$ implies $\mathcal{M} \Rightarrow \text{unl}(\overline{\mathcal{M}}')$.

We now define the live actions of a labeled PAM term as the live actions of a corresponding π -calculus term given by a translation function $\llbracket \cdot \rrbracket^r$. Intuitively, $\{l, l'\}$ is a live action whenever there are two matching prefixed processes somewhere in the PAM term that could potentially react.

Definition 5 (Translation from PAM to π -calculus). We define a function $\llbracket \cdot \rrbracket^r$ from LM to LP. $\llbracket \cdot \rrbracket^r$ is defined inductively on the structure of a labeled PAM term $\overline{\mathcal{M}}$.

$$\begin{aligned}
\llbracket \langle \overline{\mathcal{P}}, \overline{\mathcal{H}}, \mathcal{N}, \mathcal{L} \rangle \rrbracket^r &= \nu \mathcal{N}. \llbracket \overline{\mathcal{P}} \rrbracket^r \mid \llbracket \overline{\mathcal{H}} \rrbracket^r, \mathcal{L} \\
\llbracket [] \rrbracket^r &= \mathbf{0} \\
\llbracket \overline{P} :: \overline{\mathcal{P}} \rrbracket^r &= \overline{P} \mid \llbracket \overline{\mathcal{P}} \rrbracket^r \\
\llbracket \{x \rightarrow \overline{\mathcal{P}}_x\}_{x \in \mathbb{N}} \rrbracket^r &= \mid_{x \in \mathbb{N}} \llbracket \overline{\mathcal{P}}_x \rrbracket^r.
\end{aligned}$$

We can define similarly $\llbracket \cdot \rrbracket^r$ for unlabeled terms.

For brevity, we write $\nu \mathcal{N}. P$, but to be rigorous the names in \mathcal{N} should be substituted by fresh π -calculus names.

Definition 6 (Live Actions). The set of live actions of a labeled PAM term $\overline{\mathcal{M}}$ is defined as

$$LA(\overline{\mathcal{M}}) = LA(\llbracket \overline{\mathcal{M}} \rrbracket^r).$$

We also define the set of labels belonging to a live action as $L(\overline{\mathcal{M}}) = \{l \in x/x \in LA(\overline{\mathcal{M}})\}$.

Lemma 6. If $LA(\overline{\mathcal{M}}) \neq \emptyset$ then $\exists \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightsquigarrow^* \vdash \overline{\mathcal{M}}'$.

The following theorem can be seen as a fairness property for the labeled abstract machine.

Theorem 1. If $\overline{\mathcal{M}}_0 \rightarrow \dots \rightarrow \overline{\mathcal{M}}_n \rightarrow \dots$ is an infinite execution then for any strictly increasing sequence $(u_n)_{n \in \mathbb{N}}$, we have $\bigcap_{n \in \mathbb{N}} L(\overline{\mathcal{M}}_{u_n}) = \emptyset$.

The proof is technical but it relies on intuitive ideas. Informally, it follows from two key properties of the abstract machine reduction system:

- If a process $\pi_l.\overline{P}$ appears in an evaluation context in the runqueue, it will eventually reach the top of the runqueue. This is proven in [Lemma 13](#).
- The heap queues are organized following a FIFO policy.

4. Correctness

From an operational point of view, the correctness of an abstract machine can be stated by relating abstract machine executions of a process P with π -calculus executions of the same process P executed by the abstract machine. The initial state of an abstract machine running P is $\langle P, \mathcal{H}_\square \rangle$, hence we introduce the following translation function.

Definition 7 (Translation From π -calculus to PAM).

$$\llbracket P \rrbracket = \langle P :: [], \mathcal{H}_\square, \emptyset \rangle \quad \llbracket \overline{P}, \mathcal{L} \rrbracket = \langle \overline{P} :: [], \mathcal{H}_\square, \emptyset, \mathcal{L} \rangle.$$

The first property we consider is the soundness of the abstract machine with respect to the calculus. Intuitively, this means that abstract machine executions correspond to valid π -calculus executions. If a machine state \mathcal{M} , corresponding to a process state P , reduces to a machine state \mathcal{M}' , then \mathcal{M}' must correspond to a process state P' where P reduces to P' . One reduction in the π -calculus may be implemented by several reductions of the abstract machine. In order to model a one-to-one correspondence, we identify two kinds of reductions. Administrative reductions are denoted by \rightsquigarrow which models structural equivalence. Communication reductions are denoted by \vdash . We will establish a correspondence between the relations $\rightsquigarrow^* \vdash$ over PAM terms and \rightarrow over π -calculus terms. For that, we define a relation $\mathcal{M} \preceq P$ to mean that P corresponds to \mathcal{M} , read \mathcal{M} implements P .

We still need to define the relation \preceq . It has to be convincing enough that it effectively relates equivalent process states and machine states. It should at least enjoy the following two properties:

- $\llbracket P \rrbracket \preceq P$.
- If $\mathcal{M} \rightsquigarrow \mathcal{M}'$ and $\mathcal{M} \preceq P$ then $\mathcal{M}' \preceq P$.

The first property follows the idea that the initial state of an abstract machine executing P is $\llbracket P \rrbracket$. The second property follows the intuition that \rightsquigarrow is a structural, or administrative, reduction and that abstract machine states still implement the same π -calculus process after such reductions. We define \preceq as the smallest relation enjoying these two properties.

Definition 8. $\mathcal{M} \preceq P \iff \llbracket P \rrbracket \rightsquigarrow^* \mathcal{M}$.

The definition of \preceq extends naturally to labeled processes.

Alternatively, we could define $\mathcal{M} \preceq P$ as $\llbracket P \rrbracket \simeq \mathcal{M}$ where \simeq is a suitable equivalence over machine terms. Since any reasonable equivalence must verify $\mathcal{M} \rightsquigarrow^* \mathcal{M}' \implies \mathcal{M} \simeq \mathcal{M}'$, our definition of \preceq provides a stronger result.

Note that we do not have a notion of observables, although it would make the correspondence relation \preceq more convincing. However, it should be straightforward to define an observation predicate on π -calculus processes and PAM terms (such as those in [1,7,11]) and show that \preceq preserves the observables.

The following lemma relates the live actions of a labeled PAM term and a labeled process it implements.

Lemma 7. If $\overline{\mathcal{M}} \preceq C$ then $LA(\overline{\mathcal{M}}) = LA(C)$.

To prove the soundness property, we use a translation function from PAM terms to π -calculus processes. Its full definition is given in the [Appendix](#). The following lemma states the properties needed for the proof of soundness.

Lemma 8. There exists a function $\llbracket \cdot \rrbracket^{-1}$ from \mathcal{M} to P such that

- $\mathcal{M} \preceq \llbracket \mathcal{M} \rrbracket^{-1}$
- $\llbracket \llbracket P \rrbracket \rrbracket^{-1} \equiv P$
- if $\mathcal{M} \rightsquigarrow \mathcal{M}'$ then $\llbracket \mathcal{M} \rrbracket^{-1} \equiv \llbracket \mathcal{M}' \rrbracket^{-1}$
- if $\mathcal{M} \vdash \mathcal{M}'$ then $\llbracket \mathcal{M} \rrbracket^{-1} \vdash \llbracket \mathcal{M}' \rrbracket^{-1}$.

This translation function allows us to build an execution of π -calculus processes starting from an execution of PAM terms such that every PAM term corresponds to an implementation of the corresponding π -calculus term. Note that the translation function $\llbracket \cdot \rrbracket^{-1}$ given in [Definition 5](#) does not satisfy the first property of [Lemma 8](#).

Theorem 2 (Soundness). If $(\mathcal{M} \rightsquigarrow^* \vdash \mathcal{M}' \wedge \mathcal{M} \preceq P)$ then $(\exists P'. P \rightarrow P' \wedge \mathcal{M}' \preceq P')$.

Proof. The theorem follows from $\llbracket P \rrbracket \rightsquigarrow^* \vdash \mathcal{M} \implies (\exists P'. P \rightarrow P' \wedge \mathcal{M} \preceq P')$ which is a consequence of [Lemma 8](#) with $P' = \llbracket \mathcal{M} \rrbracket^{-1}$. \square

This property is not sufficient to prove the correctness of the abstract machine. Other properties are needed to characterize which executions of the π -calculus are actually implemented. First, a liveness property ensures that a PAM term is never blocked when it corresponds to a π -calculus term that can reduce.

Theorem 3 (Liveness). *If $P \rightarrow P' \wedge \mathcal{M} \leq P$ then $\exists \mathcal{M}'. \mathcal{M} \rightsquigarrow^* \mathcal{M}'$.*

Proof. We first prove: $P \rightarrow P' \implies \exists \mathcal{M}. \llbracket P \rrbracket \rightsquigarrow^* \mathcal{M}$. If $P \rightarrow P'$, we have $C \rightarrow C'$ with $\text{unl}(C) = P$ and $\text{unl}(C') = P'$, by Propositions 2 and 1. Moreover, by Lemma 7, $LA(\llbracket C \rrbracket) = LA(C)$ with $LA(C) \neq \emptyset$, by Lemma 3. We deduce $\llbracket C \rrbracket \rightsquigarrow^* \mathcal{M}'$ for some \mathcal{M}' , by Lemma 6. We conclude, by Proposition 3, that $\llbracket P \rrbracket = \text{unl}(\llbracket C \rrbracket) \rightsquigarrow^* \text{unl}(\mathcal{M}')$.

We know now that $\llbracket P \rrbracket \rightsquigarrow^* \mathcal{M}'' \mapsto \mathcal{M}'$ for some \mathcal{M}'' and \mathcal{M}' . Moreover we have $\llbracket P \rrbracket \rightsquigarrow^* \mathcal{M}$, by definition of $\mathcal{M} \leq P$. Because \rightarrow is deterministic, we conclude $\mathcal{M} \rightsquigarrow^* \llbracket P \rrbracket \rightsquigarrow^* \mathcal{M}'' \mapsto \mathcal{M}'$. \square

Finally, our main result is a fairness theorem.

Theorem 4 (Fairness). *If $\mathcal{M}_0 \rightsquigarrow^* \dots \rightsquigarrow^* \mathcal{M}_n \rightsquigarrow^* \dots$ is an infinite execution then there exists a fair execution $P_0 \rightarrow \dots \rightarrow P_n \rightarrow \dots$ such that $\mathcal{M}_i \leq P_i$ for all i .*

Proof. Let $\mathcal{M}_0 \rightsquigarrow^* \dots \rightsquigarrow^* \mathcal{M}_n \rightsquigarrow^* \dots$ be an infinite execution. we have an execution $\overline{\mathcal{M}}_0 \rightsquigarrow^* \dots \rightsquigarrow^* \overline{\mathcal{M}}_n \rightsquigarrow^* \dots$ such that for all i , $\text{unl}(\overline{\mathcal{M}}_i) = \mathcal{M}_i$, by Proposition 3.

The soundness theorem (Theorem 2) extends to the labeled calculus and gives us an execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ such that

$$\begin{array}{ccccccc} \overline{\mathcal{M}}_0 & \xrightarrow{\quad} & \overline{\mathcal{M}}_1 & \xrightarrow{\quad} & \overline{\mathcal{M}}_2 & \xrightarrow{\quad} & \dots \\ \downarrow \rightsquigarrow^* & & \downarrow \rightsquigarrow^* & & \downarrow \rightsquigarrow^* & & \\ C_0 & \xrightarrow{\quad} & C_1 & \xrightarrow{\quad} & C_2 & \xrightarrow{\quad} & \dots \end{array}$$

From Lemma 7, we have $LA(C_i) = LA(\overline{\mathcal{M}}_i)$ for all i . Then we deduce from Theorem 1 that the execution $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ is fair. By erasing the labels in both executions, we deduce the result.

$$\begin{array}{ccccccc} \text{unl}(\overline{\mathcal{M}}_0) & \xrightarrow{\quad} & \text{unl}(\overline{\mathcal{M}}_1) & \xrightarrow{\quad} & \text{unl}(\overline{\mathcal{M}}_2) & \xrightarrow{\quad} & \dots \\ \downarrow \rightsquigarrow^* & & \downarrow \rightsquigarrow^* & & \downarrow \rightsquigarrow^* & & \\ \text{unl}(C_0) & \xrightarrow{\quad} & \text{unl}(C_1) & \xrightarrow{\quad} & \text{unl}(C_2) & \xrightarrow{\quad} & \dots \end{array} \quad \square$$

5. Related work

Comparison with Pict. Correctness results in [19] include a soundness and a liveness property based on the translation function $\llbracket \cdot \rrbracket^r$ from PAM terms to π -calculus terms given in Definition 6:

- (i) $\mathcal{M} \rightarrow \mathcal{M}' \implies \llbracket \mathcal{M} \rrbracket^r \equiv \llbracket \mathcal{M}' \rrbracket^r \vee \llbracket \mathcal{M} \rrbracket^r \rightarrow \llbracket \mathcal{M}' \rrbracket^r$
- (ii) $P \rightarrow P' \implies \exists \mathcal{M}. \llbracket P \rrbracket \rightarrow \mathcal{M}$.

However, these properties are not sufficient for proving soundness or liveness. The first property means that we can build a π -calculus reduction from a PAM reduction, but does not prove that PAM reductions implement π -calculus reductions. A property relating \mathcal{M} and $\llbracket \mathcal{M} \rrbracket^r$, such as our Lemma 8 is missing. The second property tells us that if P reduces to P' , there is a PAM reduction $\llbracket P \rrbracket \rightarrow \mathcal{M}$. However, the property cannot be applied on more than the first step of execution, as we do not know if there is P'' such that $P \rightarrow P''$ and $\llbracket P'' \rrbracket = \mathcal{M}$.

In [18], the Pict abstract machine is proven correct using a notion of testing, and a realistic model of the interactions between the abstract machine and its environment. However, they do not consider fairness issues.

Fairness. Fairness has been defined using labels in CCS [3] and in the π -calculus [4,2]. We essentially followed the same idea but our presentation is simpler as we annotate labeled terms with a set of labels that allow us to generate fresh labels in the replication rules, without relying on a structured labeling language.

In [13], fairness is defined for the π -calculus by considering *normal* reductions where α -equivalence is restricted and tags similar to labels are used to distinguish processes. Fresh tags are generated using the π -calculus name restriction operator.

Correctness of abstract machine. There have been several recent papers devoted to the formal description of implementations of process calculi based on the π -calculus or the Ambient calculus. In addition to Pict, one can notably cite the Jocaml distributed implementation of the Join calculus [6,5], the Join calculus implementation of Mobile Ambients [7], Nomadic Pict [22,20], the abstract machine for the M-calculus [10], the Fusion Machine [8], the PAN and GCPAN abstract machines for Safe Ambients [11,12], the CAM abstract machine for Channel Ambients [15] and the abstract machine for the Kell calculus [1]. Most of these works [7,22,20,10,8,11,12,15,1] deal with distributed implementations of calculi, rather than local

implementation of concurrent processes like in Pict. They are defined by a translation to a low-level calculus or abstract machine. Their correctness is proven in terms of bisimilarity that does not apply to our setting, since Pict implementation makes deterministic choice and PAM reductions do not match all π -calculus reductions. Implementations that consider scheduling of processes are given in [15,10]. In [15], a soundness property is given, similar to the one given for Pict in [19]. In [10], scheduling of processes is done as in Pict using FIFO lists, but no proof of correctness is given.

6. Conclusion

In this paper, we first defined strong fairness in the π -calculus. We then proved that Pict abstract machine executions are sound with respect to π -calculus executions and that they enjoy fairness and liveness properties. These correctness results for Pict are new and in particular, fairness has not been proven for any implementation of process calculi based on the π -calculus. We believe that these techniques are simple and general enough to be adapted to other calculi.

Very little work has been done on the scheduling of processes in the π -calculus or its variants. For future research, we will investigate alternative scheduling strategies. In particular, we would like to extend Pict and its implementation with priority constraints. Processes could be prioritized in order to allocate more processor time to more important processes. In Pict, even though executions are strongly fair, in a term $P \mid Q$, P can monopolize the processor usage by spawning new subprocesses much faster than Q . One can imagine annotated processes like in $P_h \mid Q_l$ where the annotations are taken into account by the scheduler. Such a scheme would fit naturally in a calculus with hierarchical localities such as [1]. For instance, a term of the form $a[b[P] \mid c[Q]]$ can be interpreted as two (possibly untrusted) agents b and c executed by a site a . The parent site a should be able to control the processor usage of the agents it is executing.

Most correctness results of the implementations of process calculi with localities concern their distributed implementation, but do not deal with the correctness of their local implementation, *i.e.* the scheduling of processes. On the other hand, Pict defines a local implementation. It would be interesting to consider correctness results combining these two approaches. We are currently investigating the proof of a refined abstract machine based on [1].

Acknowledgments

We are grateful to Healfdene Goguen, Benjamin Pierce, Alan Schmitt and Jean-Bernard Stefani, as well as the anonymous reviewers, for their comments on earlier drafts. We thank Pablo Garralda, whose PhD thesis work inspired us to use labeled processes to study fairness [9].

Appendix. Proofs

A.1. Labeled π -calculus

Proof of Lemma 1. For the first part of the Lemma, we prove by induction on the derivation of $C \equiv D$ that:

- $C \equiv D$ and $\text{wf}(C)$ implies $\text{wf}(D)$
- $C \equiv D$ and $\text{wf}(D)$ implies $\text{wf}(C)$.

All cases are immediate. We prove the second part of the lemma by induction on the derivation of $\bar{P}, \mathcal{L} \rightarrow \bar{P}', \mathcal{L}'$.

- LR.RED We have $\text{wf}(x!(y)_l.\bar{P} \mid x?(z)_l.\bar{P}', \mathcal{L})$. By rules WF.PAR and WF.PREFIX, we have $\text{lab}(\bar{P}) \uplus \text{lab}(\bar{P}') \subseteq \mathcal{L}$. We conclude by rule WF.PAR by remarking that lab is stable by renaming.
- LR.REP As before, we can show that $\text{lab}(\bar{P}) \uplus \text{lab}(\bar{Q}\{z/y\}) \subseteq \mathcal{L}$. By the premise of the rule, we have $\text{lab}(\bar{P}) \uplus \text{lab}(\bar{Q}\{z/y\}) \uplus \text{lab}(\alpha(*x?(z)_l.\bar{Q})) \subseteq \mathcal{L}'$. We conclude with rule WF.PAR.
- LR.STR By induction and the first part of the lemma.
- LR.CTX By induction on the structure of \bar{E} . \square

Proof of Lemma 2. (i) Immediate by induction on the derivation of $\text{wf}(C)$.

- (ii) We suppose first that $\bar{P}, \mathcal{L} \rightarrow \bar{P}', \mathcal{L}' \rightarrow \bar{P}'', \mathcal{L}''$ with $l \in \text{lab}(\bar{P}'') \cap \text{lab}(\bar{P})$. Suppose $l \notin \text{lab}(\bar{P}')$, we can show that $l \in \beta(\mathcal{L}')$ for an injective function β such that $\beta(\mathcal{L}') \uplus \mathcal{L}' = \mathcal{L}''$. Besides, we have $l \in \mathcal{L} \subseteq \mathcal{L}'$ which contradicts the disjointness of $\beta(\mathcal{L}')$ and \mathcal{L}' . The same reasoning applies for the general case. \square

Proof of Proposition 1. (i) Immediate by induction on the derivation of $P \rightarrow P'$ since for all inference rules defining \rightarrow , we have a corresponding labeled rule. For rule L.STRUCT, we need the auxiliary lemma

$$(P \equiv P' \wedge P = \text{unl}(\bar{P})) \implies (\exists \bar{P}'. \bar{P} \equiv \bar{P}' \wedge P' = \text{unl}(\bar{P}'))$$

that we prove similarly by induction on the derivation of $P \equiv P'$ (we consider both directions of \equiv to handle the symmetry of \equiv).

- (ii) Immediate by induction on the derivation of $\bar{P} \rightarrow \bar{P}'$ since erasing labels in the labeled rule gives the corresponding unlabeled rule. For rule LR.STRUCT, we need to remark that $\bar{P} \equiv \bar{P}' \implies \text{unl}(\bar{P}) \equiv \text{unl}(\bar{P}')$ (by erasing the labels in the derivation of $\bar{P} \equiv \bar{P}'$). \square

Proof of Proposition 2. Immediate, any injective labeling of the prefixes works. \square

Proof of Lemma 3. We prove the direct implication by induction on the derivation of $C \rightarrow C'$. Cases LR.RED and LR.REP are immediate. Case LR.STR is a consequence of Lemma 1 and the induction hypothesis. We prove Case LR.CTX by induction on the structure of the context. The converse implication is immediate. \square

Proof of Lemma 4. The implication (ii) \implies (i) follows from the existence of a labeled execution (Propositions 1 and 2). Let us prove the converse implication. Suppose that $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ and $C'_0 \rightarrow \dots \rightarrow C'_n \rightarrow \dots$ are such that for all i , $\text{unl}(C_i) = \text{unl}(C'_i) = P_i$, $C_i = \bar{P}_i, \mathcal{L}_i$ and $C'_i = \bar{P}'_i, \mathcal{L}'_i$. We can show that there is a substitution of labels σ such that for all i , $\bar{P}'_i \sigma = \bar{P}_i$. Suppose now that $C_0 \rightarrow \dots \rightarrow C_n \rightarrow \dots$ is fair. Let $(u_n)_{n \in \mathbb{N}}$ be an increasing sequence of integers. We have $\bigcap_{n \in \mathbb{N}} L(C'_{u_n}) = \bigcap_{n \in \mathbb{N}} L(C_{u_n})\sigma = (\bigcap_{n \in \mathbb{N}} L(C_{u_n}))\sigma = \emptyset\sigma = \emptyset$. Hence, $C'_0 \rightarrow \dots \rightarrow C'_n \rightarrow \dots$ is fair. \square

A.2. Labeled PAM

Proof of Lemma 5. Immediate, similar to the proof of 1. \square

Lemma 9 (Determinism). If $\bar{\mathcal{M}} \rightarrow^n \bar{\mathcal{M}}'$ and $\bar{\mathcal{M}} \rightarrow^n \bar{\mathcal{M}}''$ then there is a bijective substitution of labels σ such that $\bar{\mathcal{M}}' \sigma = \bar{\mathcal{M}}''$ and $\text{lab}(\bar{\mathcal{M}})$ is invariant by σ .

Proof. We show it for $n = 1$ (it generalizes trivially by induction for an arbitrary n) and reason by cases on the rule used to infer both reductions (it is the same rule for both reductions as the rule is uniquely determined by the form of $\bar{\mathcal{M}}$). The interesting rules are AM.RCom1 and AM.RCom2 because the premises allow for non-determinism. Consider Case AM.RCom1, we have

$$\begin{aligned}\bar{\mathcal{M}} &= \langle *x?(y)_i.\bar{P} :: \bar{\mathcal{Q}}, \bar{\mathcal{H}}, \mathcal{L} \rangle \\ \bar{\mathcal{M}}' &= \langle \alpha(*x?(y)_i.\bar{P}) :: \bar{\mathcal{Q}}, \bar{\mathcal{H}}', \mathcal{L}' \rangle \\ \bar{\mathcal{M}}'' &= \langle \beta(*x?(y)_i.\bar{P}) :: \bar{\mathcal{Q}}, \bar{\mathcal{H}}', \mathcal{L}'' \rangle\end{aligned}$$

where α and β are injective functions and $\mathcal{L}' = \mathcal{L} \uplus \alpha(\mathcal{L})$ and $\mathcal{L}'' = \mathcal{L} \uplus \beta(\mathcal{L})$. We define σ as follows:

$$\begin{aligned}x \in \alpha(\mathcal{L}) \setminus (\alpha(\mathcal{L}) \cap \beta(\mathcal{L})) &\implies \sigma(x) = \beta(\alpha^{-1}(x)) \\ x \in \beta(\mathcal{L}) \setminus (\alpha(\mathcal{L}) \cap \beta(\mathcal{L})) &\implies \sigma(x) = \alpha(\beta^{-1}(x)) \\ \text{otherwise } \sigma(x) &= x.\end{aligned}$$

We see easily that σ is bijective and $\bar{\mathcal{M}}' \sigma = \bar{\mathcal{M}}''$. \square

Proof of Proposition 3. The first part is immediate by cases on the derivation of $\bar{\mathcal{M}} \Rightarrow \bar{\mathcal{M}}'$. We simply need to notice that for every unlabeled rule there is a corresponding labeled rule. In particular, in rules AM.RCom1 and AM.RCom2, we can find α such that the premises hold.

The second part is by cases on the derivation of $\bar{\mathcal{M}} \Rightarrow \bar{\mathcal{M}}'$. We simply need to erase the labels in the rule used to infer the corresponding unlabeled reduction. \square

Lemma 10 (Uniqueness of Labeling). (i) If $\bar{\mathcal{M}} \in \text{WFM}$ then no label l occurs more than once in $\bar{\mathcal{M}}$.

(ii) If $\bar{\mathcal{M}} \in \text{WFM}$, $\bar{\mathcal{M}} \rightarrow^* \bar{\mathcal{M}}' \rightarrow^* \bar{\mathcal{M}}''$ and $l \in \text{lab}(\bar{\mathcal{M}}) \cap \text{lab}(\bar{\mathcal{M}}'')$, then $l \in \text{lab}(\bar{\mathcal{M}}')$.

Proof. The proof is similar to the corresponding proof for the labeled π -calculus. \square

Lemma 11. If $\bar{\mathcal{M}} \rightsquigarrow \bar{\mathcal{M}}'$ then $\text{LA}(\bar{\mathcal{M}}) = \text{LA}(\bar{\mathcal{M}}')$.

Proof. The two following properties are direct consequences of the definition of LA :

- $C \equiv D \implies \text{LA}(C) = \text{LA}(D)$
- $\mathcal{L} \subseteq \mathcal{L}' \implies \text{LA}(\bar{P}, \mathcal{L}) = \text{LA}(\bar{P}, \mathcal{L}')$.

If $\llbracket \bar{\mathcal{M}} \rrbracket^r = \bar{P}, \mathcal{L}$ and $\llbracket \bar{\mathcal{M}}' \rrbracket^r = \bar{P}', \mathcal{L}'$, we can show by cases on the derivation of $\bar{\mathcal{M}} \rightsquigarrow \bar{\mathcal{M}}'$ that $\llbracket \bar{P}, \mathcal{L} \rrbracket^r \equiv \llbracket \bar{P}', \mathcal{L}' \rrbracket^r$ (notice that two structurally equivalent processes must have the same set of labels). We then have

$$\text{LA}(\bar{\mathcal{M}}) = \text{LA}(\bar{P}, \mathcal{L}) = \text{LA}(\bar{P}, \mathcal{L}') = \text{LA}(\bar{P}', \mathcal{L}') = \text{LA}(\bar{\mathcal{M}}'). \quad \square$$

Definition 9 (Size of a Process). We define the size of a process and of a process queue.

$$\begin{aligned}
 \text{size}(\mathbf{0}) &= 1 \\
 \text{size}(\pi.P) &= 1 \\
 \text{size}(\nu x.P) &= \text{size}(P) + 1 \\
 \text{size}(P \mid P') &= \text{size}(P) + \text{size}(P') + 1 \\
 \text{size}([]) &= 0 \\
 \text{size}(P :: \mathcal{P}) &= \text{size}(P) + \text{size}(\mathcal{P}).
 \end{aligned}$$

This definition extends to labeled processes and labeled Pict terms.

Proof of Lemma 6. We will prove the contrapositive proposition. Suppose that for any n and $\overline{\mathcal{M}}'$ such that $\overline{\mathcal{M}} \rightsquigarrow^n \overline{\mathcal{M}}'$ there is no $\overline{\mathcal{M}}''$ such that $\overline{\mathcal{M}}' \mapsto \overline{\mathcal{M}}''$.

We show first that $\overline{\mathcal{M}} \rightsquigarrow^* \overline{\mathcal{M}}'$ with $\overline{\mathcal{M}}' = \langle [], \overline{\mathcal{H}}' \rangle$. Suppose $\overline{\mathcal{M}} = \langle \overline{\mathcal{P}}, \overline{\mathcal{H}} \rangle$, we prove by induction on $n = \text{size}(\overline{\mathcal{P}})$ that there exists $p \leq n$ such that $\overline{\mathcal{M}} \rightsquigarrow^p \overline{\mathcal{M}}'$ with $\overline{\mathcal{M}}' = \langle [], \overline{\mathcal{H}}' \rangle$. If $n = 0$, $\overline{\mathcal{P}} = []$ which proves the result. If $n > 0$, $\overline{\mathcal{P}} \neq []$ and we have either $\overline{\mathcal{M}} \rightsquigarrow \overline{\mathcal{M}}'$ or $\overline{\mathcal{M}} \mapsto \overline{\mathcal{M}}'$. By hypothesis, only the first case is possible. Moreover if $\overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$, we remark that $\text{size}(\overline{\mathcal{P}}') < \text{size}(\overline{\mathcal{P}})$. We can conclude by applying the induction hypothesis.

By Lemma 11, we have $LA(\overline{\mathcal{M}}) = LA(\overline{\mathcal{M}}')$. We conclude by remarking that $LA(\overline{\mathcal{M}}') = \emptyset$ since in the heap, the process queue for a given name cannot contain two complementary actions (cf. Remark 1). \square

Lemma 12. If $l \in L(\langle \overline{\mathcal{P}}, \overline{\mathcal{H}}, \mathcal{N}, \mathcal{L} \rangle)$, we have either $\overline{\mathcal{P}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$ for some $\overline{\mathbf{E}}, \overline{\mathcal{P}}_0$ and $\overline{\mathcal{P}}_1$ or $\overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_l.P :: \overline{\mathcal{Q}}'$ for some $x, \overline{\mathcal{Q}}, \overline{\mathcal{Q}}'$ and $\pi_l.P$.

Proof. Let us write $\overline{\mathcal{M}} = \langle \overline{\mathcal{P}}, \overline{\mathcal{H}}, \mathcal{N}, \mathcal{L} \rangle$. We define $\overline{\mathcal{P}}_0 = \llbracket \overline{\mathcal{P}} \rrbracket' \mid \llbracket \overline{\mathcal{H}} \rrbracket'$. We have $\llbracket \overline{\mathcal{M}} \rrbracket' = \nu \mathcal{N}. \llbracket \mathcal{P} \rrbracket' \mid \llbracket \overline{\mathcal{H}} \rrbracket', \mathcal{L}$. From the definition of LA , we see that $LA(\overline{\mathcal{M}}) = LA(\llbracket \overline{\mathcal{M}} \rrbracket') = LA(\overline{\mathcal{P}}_0, \mathcal{L})$. Moreover, for some $l', \overline{\mathcal{P}}_0, \mathcal{L} \equiv \nu \tilde{x}. \pi_l.Q \mid \pi_{l'}.\overline{\mathcal{Q}}' \mid \overline{\mathcal{Q}}'', \mathcal{L}$ where π_l and $\pi_{l'}$ are matching prefixes. Since $\overline{\mathcal{M}}$ is well-formed, we know that either $l \in \text{lab}(\overline{\mathcal{P}})$ or $l \in \text{lab}(\overline{\mathcal{H}})$.

In the first case, $\overline{\mathcal{P}} = \overline{\mathcal{P}}_0 :: \overline{\mathcal{P}} :: \overline{\mathcal{P}}_1$ and we have $\overline{\mathcal{P}}_0 = \overline{\mathcal{P}} \mid \overline{\mathcal{Q}}$ with $l \in \text{lab}(\overline{\mathcal{P}})$, for some $\overline{\mathcal{Q}}$. We deduce $\overline{\mathcal{P}} = \overline{\mathbf{E}}[\pi_l.Q] \text{ for some } \overline{\mathbf{E}}$.

In the second case, $\overline{\mathcal{H}} = \overline{\mathcal{H}}' \oplus \{x \rightarrow \overline{\mathcal{P}}_0 :: \pi_{l'}.\overline{\mathcal{P}} :: \overline{\mathcal{P}}_1\}$ and we have $\overline{\mathcal{P}}_0 = \pi_{l'}.\overline{\mathcal{P}} \mid \overline{\mathcal{Q}}$ for some $\overline{\mathcal{Q}}$ and $l \in \text{lab}(\pi_{l'}.\overline{\mathcal{P}})$. We deduce $l' = l$. \square

The following lemma states a key property of the abstract machine reduction system. If a process $\pi_l.P$ appears in an evaluation context in the runqueue, it will eventually reach the top of the runqueue.

Lemma 13. If $\overline{\mathcal{M}} = \langle \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1, \overline{\mathcal{H}} \rangle$ then $\overline{\mathcal{M}} \rightarrow^* \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$ with $\overline{\mathcal{P}}' = \pi_l.P :: \overline{\mathcal{P}}'_0$.

Proof. We give first an informal proof. If the runqueue is of the form $\overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$, after a finite number (usually one) of reduction steps, the size of the term $\overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[.]$ strictly decreases (for an appropriate definition of size). Eventually, the size reaches \perp which means $\pi_l.P$ is at the top of the runqueue. In rules AM.NIL as well as the three “push” rules, the first process of the queue is dropped. In rules AM.NEW, AM.PAR and all the communication rules besides AM.RCOM1, the first process is replaced by a smaller process. In the rule AM.RCOM1, the first process of the queue (a replicated receiver) is dropped after a number of steps corresponding to the number of processes in the heap that can react with it.

Formally, we first define a distance function $d(., .)$. If $\overline{\mathcal{P}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$ then $d(l, \overline{\mathcal{P}}) = (\text{size}(\overline{\mathcal{P}}_0), \text{size}(\overline{\mathbf{E}}[\pi_l.P]))$. Otherwise, $d(l, \overline{\mathcal{P}}) = \perp$. This definition is correct because such a decomposition of $\overline{\mathcal{P}}$ is unique. The lemma follows from these three properties:

- (i) If $d(l, \overline{\mathcal{P}}) = (u, v)$ then $u \geq 0$ and $v \geq 1$.
- (ii) $d(l, \overline{\mathcal{P}}) = (0, 1)$ if and only if $\overline{\mathcal{P}} = \pi_l.P :: \overline{\mathcal{P}}_0$.
- (iii) If $d(l, \overline{\mathcal{M}}) = (u, v)$ with $(u, v) \neq (0, 1)$ then $\exists \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}'$ with $d(l, \overline{\mathcal{M}}') = (u', v')$ such that $v' < v$, or $u' < u$ and $v' = v$.

The first two properties are immediate. We prove the third one. First, notice that if $d(l, \overline{\mathcal{M}}) \neq \perp$, then $\exists \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightarrow \overline{\mathcal{M}}'$. We reason by cases on the reduction rule used.

AM.NIL We have $\overline{\mathcal{P}} = \mathbf{0} :: \overline{\mathcal{Q}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$. $\overline{\mathcal{M}}$ reduces to $\overline{\mathcal{M}}'$ with $\overline{\mathcal{P}}' = \overline{\mathcal{Q}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$. We deduce that $u' = u - 1 < u$ and $v' = v$.

AM.PAR We have $\overline{\mathcal{P}} = (\overline{\mathcal{P}} \mid \overline{\mathcal{Q}}) :: \overline{\mathcal{Q}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l.P] :: \overline{\mathcal{P}}_1$. $\overline{\mathcal{M}}$ reduces to $\overline{\mathcal{M}}'$ with $\overline{\mathcal{P}}' = \overline{\mathcal{P}} :: \overline{\mathcal{Q}} :: \overline{\mathcal{Q}}$. If $\overline{\mathcal{P}}_0 \neq []$ then $u' = u - \text{size}(\overline{\mathcal{Q}}) - 1 < u$ and $v' = v$. If $\overline{\mathcal{P}}_0 = []$ then we have either $\overline{\mathbf{E}}[\pi_l.P] = \overline{\mathbf{E}}'[\pi_l.P] \mid \overline{\mathcal{P}}'$ or $\overline{\mathbf{E}}[\pi_l.P] = \overline{\mathcal{P}}' \mid \overline{\mathbf{E}}'[\pi_l.P]$. In both cases, $v' = v - \text{size}(\overline{\mathcal{P}}') - 1 < v$.

AM.NEW Similar to the previous case, using the fact that $|P| = |P\{y/x\}|$.

AM.PUSHRECEIVER, AM.PUSHMESSAGE, AM.PUSHREPRECEIVER These three cases are identical. We have $\overline{\mathcal{P}} = \pi'_{l'}. \overline{P}' :: \overline{\mathcal{Q}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l. \overline{P}] :: \overline{\mathcal{P}}_1$. Necessarily, $\overline{\mathcal{P}}_0 \neq []$. $\overline{\mathcal{P}}' = \overline{\mathcal{Q}} = \overline{\mathcal{P}}'_0 :: \overline{\mathbf{E}}[\pi_l. \overline{P}] :: \overline{\mathcal{P}}_1$. We deduce that $u' = u - 1 < u$ and $v' = v$.

AM.COM1, AM.COM2, AM.RCOM2 These three cases are similar to the previous case.

AM.RCOM1 In this case, $\overline{\mathcal{P}} = *x?(y). \overline{P}' :: \overline{\mathcal{Q}}$. After a finite number of reductions (corresponding to the number of terms in $\overline{\mathcal{H}}(x)$), we have $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}'$ with $\overline{\mathcal{P}}' = \overline{\mathcal{Q}}$. We conclude as for the other communication rules. \square

Lemma 14. Let us suppose $\overline{\mathcal{M}} = \langle \overline{\mathcal{P}}, \overline{\mathcal{H}} \rangle$

- (i) If $l \in L(\overline{\mathcal{M}})$ then we have $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$ for some $\overline{\mathcal{M}}'$ and one of these two conditions holds:
 - $l \notin \text{lab}(\overline{\mathcal{M}}')$.
 - There is x such that $\overline{\mathcal{H}}(x)$ is of the form $\overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$.
- (ii) If for some y , $\overline{\mathcal{H}}(y) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$ then for all $\overline{\mathcal{M}}'$ such that $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$ one of these two conditions holds:
 - $l \notin \text{lab}(\overline{\mathcal{M}}')$.
 - $\overline{\mathcal{H}}'(y) = \overline{\mathcal{Q}}_0 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_0$ for some $\overline{\mathcal{Q}}_0, \overline{\mathcal{Q}}'_0$ with $|\overline{\mathcal{Q}}_0| \leq |\overline{\mathcal{Q}}|$.
- (iii) If for some y , $\overline{\mathcal{H}}(y) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$ and $l \in L(\overline{\mathcal{M}})$ then there is $\overline{\mathcal{M}}'$ such that $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$ and one of these two conditions holds
 - $l \notin \text{lab}(\overline{\mathcal{M}}')$.
 - $\overline{\mathcal{H}}'(y) = \overline{\mathcal{Q}}_0 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_0$ for some $\overline{\mathcal{Q}}_0, \overline{\mathcal{Q}}'_0$ with $|\overline{\mathcal{Q}}_0| < |\overline{\mathcal{Q}}|$.
- (iv) If for some x , $\overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$ then there is $\overline{\mathcal{M}}'$ such that $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}'$ and $\forall \overline{\mathcal{M}}''. \overline{\mathcal{M}}' \rightarrow^* \overline{\mathcal{M}}'' \implies l \notin L(\overline{\mathcal{M}}'')$.

Proof. (i) If $l \in L(\overline{\mathcal{M}})$, by Lemma 12 we have either $\overline{\mathcal{P}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_l. \overline{P}] :: \overline{\mathcal{P}}_1$ for some context $\overline{\mathbf{E}}$ or $\overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$ for some $x, \overline{\mathcal{Q}}$ and $\overline{\mathcal{Q}}'$. In the second case, the lemma is proved with $\overline{\mathcal{M}}' = \overline{\mathcal{M}}$. In the first case, we apply Lemma 13 and we have $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}_0$ with $\overline{\mathcal{M}}_0 = \langle \pi_l. \overline{P} :: \overline{\mathcal{P}}'_0, \overline{\mathcal{H}}_0 \rangle$. We then have $\overline{\mathcal{M}}_0 \rightarrow \overline{\mathcal{M}}'$. If this reduction is inferred by a communication rule, then $l \notin \text{lab}(\overline{\mathcal{M}}')$, otherwise it is inferred by a pushing rule and $\overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle$ with $\overline{\mathcal{H}}' = \overline{\mathcal{H}}_0 :: \pi_l. \overline{P}$.

(ii) We prove it by induction on n in $\overline{\mathcal{M}} \rightarrow^n \overline{\mathcal{M}}'$. If $n = 0$, the result is immediate. If $n \geq 0$, we have $\overline{\mathcal{M}} \rightarrow \overline{\mathcal{M}}' \rightarrow^n \overline{\mathcal{M}}''$. Consider now all the possible rules to derive $\overline{\mathcal{M}} \rightarrow \overline{\mathcal{M}}'$. In the rules AM.NIL, AM.PAR and AM.RES, we have $\overline{\mathcal{H}}(y) = \overline{\mathcal{H}}(x)$. In the rules AM.PUSHMESSAGE, AM.PUSHRECEIVER and AM.PUSHREPRECEIVER, if $x \neq y$, we have $\overline{\mathcal{H}}'(y) = \overline{\mathcal{H}}(y)$. Otherwise, $\overline{\mathcal{H}}'(y) = \overline{\mathcal{H}}(y) :: P$. In the rules AM.COM1, AM.COM2, AM.RCOM1 and AM.RCOM2, if $x = y$ then $\overline{\mathcal{H}}'(y) = \overline{\mathcal{H}}(y)$. If $x \neq y$, either $l \notin \text{lab}(\overline{\mathcal{M}}')$ or $\overline{\mathcal{H}}'(y) = \overline{\mathcal{Q}}_0 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_0$ with $|\overline{\mathcal{Q}}_0| \leq |\overline{\mathcal{Q}}|$. In the first case, the result comes from Lemma 10. In the second case, we conclude by the induction hypothesis.

(iii) We have $\{l, l'\} \in LA(\overline{\mathcal{M}})$ for some l' . Then by Lemma 12 we have either $\overline{\mathcal{P}} = \overline{\mathcal{P}}_0 :: \overline{\mathbf{E}}[\pi_{l'}. \overline{P}'] :: \overline{\mathcal{P}}'_0$ for some context $\overline{\mathbf{E}}$ or $\overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_{l'}. \overline{P}' :: \overline{\mathcal{Q}}'$ for some $x, \overline{\mathcal{Q}}$ and $\overline{\mathcal{Q}}'$. We can show that the second case contradicts the hypothesis $\{l, l'\} \in LA(\overline{\mathcal{M}})$, so only the first case is possible. By Lemma 13, we have $\overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}_0$ with $\overline{\mathcal{M}}_0 = \langle \pi_{l'}. \overline{P}' :: \overline{\mathcal{P}}'_0, \overline{\mathcal{H}}_0 \rangle$. By (ii), we have $l \notin \text{lab}(\overline{\mathcal{M}}_0)$ or $\overline{\mathcal{H}}_0(x) = \overline{\mathcal{Q}}_0 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_0$ with $|\overline{\mathcal{Q}}_0| \leq |\overline{\mathcal{Q}}|$. In the first case, the result is proven with $\overline{\mathcal{M}}' = \overline{\mathcal{M}}_0$. In the second case, we can show that $\{l, l'\} \in LA(\overline{\mathcal{M}}_0)$ and that $\overline{\mathcal{M}}_0 \rightarrow \overline{\mathcal{M}}_1$ for some $\overline{\mathcal{M}}_1$, where \rightarrow is inferred by a communication rule on y . If $|\overline{\mathcal{Q}}_0| = 0$, then $l \notin \text{lab}(\overline{\mathcal{M}}_1)$. If $|\overline{\mathcal{Q}}_0| > 0$, then $\overline{\mathcal{M}}_1 = \langle \overline{\mathcal{P}}_1, \overline{\mathcal{H}}_1 \rangle$ with $\overline{\mathcal{H}}_1 = \overline{\mathcal{Q}}_1 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_1$ with $|\overline{\mathcal{Q}}_1| < |\overline{\mathcal{Q}}_0|$. In these two cases, the result is proven with $\overline{\mathcal{M}}' = \overline{\mathcal{M}}_1$.

(iv) We define $I(k)$:

$$I(k) \equiv \overline{\mathcal{M}} = \langle \overline{\mathcal{P}}, \overline{\mathcal{H}} \rangle \wedge \exists x. \overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}' \wedge |\overline{\mathcal{Q}}| = k \\ \implies \exists \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' \wedge (\overline{\mathcal{M}}' \rightarrow^* \overline{\mathcal{M}}'' \implies l \notin L(\overline{\mathcal{M}}'')).$$

We show $\forall k \geq 0. I(k)$ by strong induction on k . We distinguish two cases. If $\forall \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' \implies l \notin L(\overline{\mathcal{M}}')$, then the theorem holds trivially. Otherwise, we have $\exists \overline{\mathcal{M}}' = \langle \overline{\mathcal{P}}', \overline{\mathcal{H}}' \rangle. \overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' \wedge l \in L(\overline{\mathcal{M}}')$. By (ii), we have $\overline{\mathcal{H}}'(x) = \overline{\mathcal{Q}}_0 :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'_0$ with $|\overline{\mathcal{Q}}_0| \leq |\overline{\mathcal{Q}}|$. We can apply (iii). In the first case, the property is proven. In the second case, we conclude by the induction hypothesis. \square

Proof of Theorem 1. Let us consider an infinite execution

$$\overline{\mathcal{M}}_1 \rightarrow \dots \rightarrow \overline{\mathcal{M}}_n \rightarrow \dots$$

where $\overline{\mathcal{M}}_k = \langle \overline{\mathcal{P}}_k, \overline{\mathcal{H}}_k \rangle$ for $k \in \mathbb{N}^*$. Let $i \in \mathbb{N}^*$ and $l \in L(\overline{\mathcal{M}}_i)$. We want to show that there is a j such that $\forall i > j. l \notin L(\overline{\mathcal{M}}_i)$.

According to Lemma 14(i), we have some $\overline{\mathcal{M}}$ such that $\overline{\mathcal{M}}_i \rightarrow^* \overline{\mathcal{M}}$. Moreover, one of these two conditions hold:

- $l \notin \text{lab}(\overline{\mathcal{M}})$.
- $\overline{\mathcal{H}}(x) = \overline{\mathcal{Q}} :: \pi_l. \overline{P} :: \overline{\mathcal{Q}}'$ for some x .

By Lemma 9, there is $j \geq i$ and a bijective substitution of labels σ such that $\overline{\mathcal{M}}\sigma = \overline{\mathcal{M}}_j$, and in particular $\sigma(l) = l$. The same two conditions hold for $\overline{\mathcal{M}}_j$.

- $l \notin \text{lab}(\overline{\mathcal{M}}_j)$.
- $\overline{\mathcal{H}}_j(x) = \overline{\mathcal{Q}} :: \pi_l.\overline{P} :: \overline{\mathcal{Q}}'$ for some x .

In the first case, we know from [Lemma 10](#) that $\forall i \geq j. l \notin \text{lab}(\overline{\mathcal{M}}_i)$. We deduce $\forall i \geq j. l \notin L(\overline{\mathcal{M}}_i)$. In the second case, we have by [Lemma 14\(iv\)](#):

$$\exists \overline{\mathcal{M}}'. \overline{\mathcal{M}} \rightarrow^* \overline{\mathcal{M}}' \text{ and } (\forall \overline{\mathcal{M}}''. \overline{\mathcal{M}}' \rightarrow^* \overline{\mathcal{M}}'' \implies l \notin L(\overline{\mathcal{M}}'')). \quad (1)$$

By [Lemma 9](#), we have $j' \geq j$ and a bijective substitution of labels σ such that $\overline{\mathcal{M}}'\sigma = \overline{\mathcal{M}}_{j'}$ and $\sigma(l) = l$. Now suppose $k \geq j'$, we have $\overline{\mathcal{M}}_{j'} \rightarrow^* \overline{\mathcal{M}}_k$. We can show that $\overline{\mathcal{M}}_{j'}\sigma^{-1} \rightarrow^* \overline{\mathcal{M}}_k\sigma^{-1}$. By (1), we deduce that $l \notin L(\overline{\mathcal{M}}_k\sigma^{-1})$ and that $l \notin L(\overline{\mathcal{M}}_k)$. We showed that $\forall k \geq j'. l \notin L(\overline{\mathcal{M}}_k)$. \square

A.3. Correctness

Proof of Lemma 7. We have $\llbracket \overline{P} \rrbracket^r = \overline{P}$, and $LA(\llbracket \overline{P} \rrbracket) = LA(\llbracket \overline{P} \rrbracket^r) = LA(\overline{P})$. We conclude by [Lemma 11](#). \square

The following definition is a function $\llbracket \cdot \rrbracket^{-1}$ from PAM terms to π -calculus processes. It will be used in the soundness theorem to build an execution of π -calculus processes starting from an execution of PAM terms such that every PAM term corresponds to an implementation of the corresponding π -calculus term.

The functions $\llbracket \cdot \rrbracket^{-1}$ and $\llbracket \cdot \rrbracket^r$ ([Definition 5](#)) are very close. In particular, we have $\llbracket \mathcal{M} \rrbracket^{-1} \equiv \llbracket \mathcal{M} \rrbracket^r$ for any machine term \mathcal{M} . However, $\llbracket \cdot \rrbracket^r$ cannot be used to prove the soundness property because it does not satisfy the first property of [Lemma 8](#).

Intuitively, this comes from the fact that $\llbracket P_1 \mid P_2 \rrbracket$ and $\llbracket P_2 \mid P_1 \rrbracket$ may behave differently in general. In particular, to have $\llbracket \llbracket \mathcal{M} \rrbracket^{-1} \rrbracket \rightsquigarrow^* \mathcal{M}$, processes in \mathcal{M} should be arranged in a proper order (with respect to parallel composition).

Definition 10. We define a function $\llbracket \cdot \rrbracket^{-1}$ from \mathbf{M} to \mathbf{P} . We suppose $\mathcal{M} = \langle \mathcal{P}_p, \mathcal{H}, \mathcal{N} \rangle$ with $\mathcal{H} = \{x_i \rightarrow \mathcal{P}_i\}_{i \in 0..p-1}$. We define $(P_i)_{i \in 0..n}$ such that

$$\mathcal{P}_0 :: \dots :: \mathcal{P}_p = P_0 :: \dots :: P_n.$$

Then we define $(Q_i)_{i \in 0..n}$ as follows:

$$\begin{aligned} Q_0 &= P_0 \\ Q_p &= Q_{p-1} \mid P_{n-p+1}, p \in \{1..n\}. \end{aligned}$$

Finally, if $\{x_1, \dots, x_k\} = \mathcal{N}$, we define:

$$\llbracket \mathcal{M} \rrbracket^{-1} = \nu a_1 \dots a_k. Q_n \{a_1/x_1, \dots, a_k/x_k\}$$

where the names a_i for $i \in \{1..k\}$ are all distinct and different from names in Q_n . We suppose also that $\forall i \in \{1..k-1\}$. $\text{freshn}(\{x_1, \dots, x_i\}) = x_{i+1}$. This assumption is justified by [Remark 1](#). Names in \mathcal{N} have been generated by rule [AM.NEW](#), hence, there is always an ordering x_1, \dots, x_k of \mathcal{N} such that $\text{freshn}(\{x_1, \dots, x_i\}) = x_{i+1}$. Alternatively, we could define $\mathcal{M} \leq P$ as $\llbracket P \rrbracket \rightsquigarrow^*_{=_{\alpha}} \mathcal{M}$ where $=_{\alpha}$ identifies machine terms that differ only in their generated names.

Proof of Lemma 8. We suppose $\mathcal{M} = \langle \mathcal{P}_p, \mathcal{H}, \mathcal{N} \rangle$ with $\mathcal{H} = \{x_i \rightarrow \mathcal{P}_i\}_{i \in 0..p-1}$. We reuse the notations of definition 10. We show that:

$$\begin{aligned} \llbracket \llbracket \mathcal{M} \rrbracket^{-1} \rrbracket &\rightsquigarrow^* \langle Q_n, \mathcal{H}_{\square}, \mathcal{N} \rangle \\ &\rightsquigarrow^* \langle \mathcal{P}_0 :: \dots :: \mathcal{P}_p, \mathcal{H}_{\square}, \mathcal{N} \rangle \\ &\rightsquigarrow^* \langle \mathcal{P}_p, \mathcal{H}, \mathcal{N} \rangle = \mathcal{M}. \end{aligned}$$

Hence, we have $\llbracket \llbracket \mathcal{M} \rrbracket^{-1} \rrbracket \rightsquigarrow^* \mathcal{M}$ and $\mathcal{M} \leq \llbracket \mathcal{M} \rrbracket^{-1}$. We detail these three steps:

- We have

$$\llbracket \llbracket \mathcal{M} \rrbracket^{-1} \rrbracket = \langle \nu a_1 \dots a_k. Q_n \{a_1/x_1, \dots, a_k/x_k\}, \mathcal{H}_{\square}, \emptyset \rangle \rightsquigarrow^* \langle Q_n, \mathcal{H}_{\square}, \mathcal{N} \rangle$$

by successive applications of rule [AM.NEW](#), and the condition over generated names $\text{freshn}(\{x_1, \dots, x_i\}) = x_{i+1}$.

- We have

$$\langle Q_n, \mathcal{H}_{\square}, \emptyset \rangle \rightsquigarrow^* \langle P_0 :: \dots :: P_n, \mathcal{H}_{\square}, \mathcal{N} \rangle$$

by successive applications of rule [AM.PAR](#).

- We have

$$\langle P_0 :: \dots :: P_n, \mathcal{H}_{\square}, \mathcal{N} \rangle \rightsquigarrow^* \langle \mathcal{P}_p, \mathcal{H}, \mathcal{N} \rangle$$

by successive applications of pushing rules ([AM.PUSHRECEIVER](#), [AM.PUSHMESSAGE](#) and [AM.PUSHREPRECEIVER](#)).

The second property $\llbracket \llbracket P \rrbracket \rrbracket^{-1} = P$ is immediate by definition of $\llbracket \cdot \rrbracket^{-1}$ and $\llbracket \cdot \rrbracket$. For the two remaining properties, we notice that $\llbracket \mathcal{M} \rrbracket^{-1} \equiv \llbracket \mathcal{M} \rrbracket^r$ so it is enough to prove the result for $\llbracket \cdot \rrbracket^r$ instead of $\llbracket \cdot \rrbracket^{-1}$. The proof is by cases on the definitions of \rightsquigarrow and \mapsto and has been given in [\[19\]](#). \square

References

- [1] Philippe Bidingier, Alan Schmitt, Jean-Bernard Stefani, An abstract machine for the Kell calculus, in: 7th IFIP International Conference on Formal Methods for Object-Based Distributed Systems, FMOODS, Athens, Greece, June 2005, pp. 43–58.
- [2] Diletta Cacciagrano, Flavio Corradini, Catuscia Palamidessi, Fairpi, in: Proceedings of EXPRESS'06, ENTCS, 2006.
- [3] Gerardo Costa, Colin Stirling, Weak and strong fairness in CCS, in: Michal Chytil, Václav Koubek (Eds.), MFCS, in: Lecture Notes in Computer Science, vol. 176, Springer, 1984, pp. 245–254.
- [4] F. Corradini, D.R. Cacciagrano, Fairness in the pi-calculus, Technical Report, Dipartimenti di Informatica, Università di L'Aquila, TR 005/2004, 2004.
- [5] Fabrice Le Fessant, JoCaml: Conception et Implantation d'un Langage à Agents Mobiles, Ph.D. Thesis, Ecole Polytechnique, 2001.
- [6] C. Fournet, G. Gonthier, J.J. Levy, L. Maranget, D. Remy, A calculus of mobile agents, in: Proceedings 7th International Conference on Concurrency Theory, CONCUR'96, in: Lecture Notes in Computer Science, vol. 1119, Springer Verlag, 1996, pp. 406–421.
- [7] C. Fournet, J.J. Levy, A. Schmitt, An asynchronous distributed implementation of mobile ambients, in: Proceedings of the International IFIP Conference TCS 2000, Sendai, Japan, in: Lecture Notes in Computer Science, vol. 1872, Springer, 2000, pp. 348–364.
- [8] Philippa Gardner, Cosimo Laneve, Lucian Wischik, The fusion machine, in: L. Brim, P. Jancar, M. Kretinsky (Eds.), CONCUR 2002, in: LNCS, vol. 2421, Springer-Verlag, 2002, pp. 418–433.
- [9] Pablo Garralda, Boxed ambients for global computing, Ph.D. Thesis, Stevens Institute of Technology, NJ, USA, 2007.
- [10] F. Germain, M. Lacoste, J.B. Stefani, An abstract machine for a higher-order distributed process calculus, in: Proceedings of the EACTS Workshop on Foundations of Wide Area Network Computing, F-WAN, July 2002.
- [11] Paola Giannini, Davide Sangiorgi, Andrea Valente, Safe ambients: Abstract machine and distributed implementation, Sci. Comput. Program. 59 (3) (2006) 209–249.
- [12] Daniel Hirschhoff, Damien Pous, Davide Sangiorgi, A correct abstract machine for safe ambients, in: COORDINATION, 2005, pp. 17–32.
- [13] Naoki Kobayashi, A type system for lock-free processes, Inform. Comput. 177 (2) (2002) 122–159.
- [14] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile processes, parts I and II, Inform. Comput. 100 (1) (1992) 1–78.
- [15] A. Phillips, N. Yoshida, S. Eisenbach, A distributed abstract machine for boxed ambient calculi, in: Proceedings of ESOP 2004, in: LNCS, Springer-Verlag, 2004.
- [16] Benjamin C. Pierce, David N. Turner, Pict: A programming language based on the pi-calculus, in: Gordon Plotkin, Colin Stirling, Mads Tofte (Eds.), Proof, Language and Interaction: Essays in Honour of Robin Milner, MIT Press, 2000, pp. 455–494.
- [17] D. Sangiorgi, D. Walker, The π -calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
- [18] Peter Sewell, On implementations and semantics of a concurrent programming language, in: Proceedings of CONCUR 97: Concurrency Theory (Warsaw), in: LNCS, vol. 1243, 1997, pp. 391–405.
- [19] D. Turner, The polymorphic π -calculus: Theory and implementation, Technical Report, University of Edinburgh, GB, 1996.
- [20] A. Unyapoth, P. Sewell, Nomadic pict: Correct communication infrastructures for mobile computation, in: Proceedings ACM Int. Conf. on Principles of Programming Languages, POPL, 2001, pp. 116–127.
- [21] Lucian Wischik, Explicit fusions: Theory and implementation, Ph.D. Thesis, Computer Laboratory, University of Cambridge, 2001.
- [22] P. Wojciechowski, P. Sewell, Nomadic pict: Language and infrastructure, IEEE Concurr. 8 (2) (2000) 42–52.